

Contenido

CARDVIEW.....	119
IMPLEMENTANDO LISTAS.....	120
Listas desplegables (Spinners).....	121
ListView	122
ListActivity.....	123
RecyclerView	124
Click en cualquier lugar de la vista	132
Otros efectos sobre el Recycler	134

7.

Interfaz de Usuario III. ListView y RecyclerView

CARDVIEW

En el SDK de Android 5.0 Lollipop, se da soporte a un elemento que aunque se utilizaba en diversas aplicaciones, tenía que usar librerías de terceros. Este nuevo componente llamado **CardView** es la implementación que nos proporciona Google del elemento visual en forma de tarjetas de información que tanto utiliza en muchas de sus aplicaciones, entre ellas Google Now.

Deberemos incluir la librería `android.support.v7.widget` a nuestro proyecto. Después tan sólo habrá que añadir a nuestro layout XML un control de tipo **<android.support.v7.widget.CardView>** y establecer algunas de sus propiedades principales.

Un **CardView** no es más que una extensión de **FrameLayout** con esquinas redondeadas y una sombra inferior, dentro de un **CardView** podemos añadir todos los controles que necesitemos. A este componente le podemos modificar el fondo o cualquier otra propiedad que sea modificable en cualquier layout, pero las características que lo diferencian del resto son las que nos permiten modificar el sombreado y el radio de las esquinas, propiedades `cardElevation` y `cardCornerRadius` respectivamente.

Vamos a crear **un nuevo proyecto EjemploLista**, en el que crearemos un nuevo layout llamado `cardview.xml` con el siguiente código, para comprobar el funcionamiento de las propiedades anteriormente nombradas:

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.v7.widget.CardView xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:id="@+id/cardview"
    app:cardElevation="10dp"
    app:cardUseCompatPadding="true"
    app:cardCornerRadius="2dp"
    app:cardBackgroundColor="@android:color/holo_green_light">
</android.support.v7.widget.CardView>
```

No son muchas más las opciones de personalización que nos ofrece **CardView**, pero con poco esfuerzo deben ser suficientes para crear tarjetas más “sofisticadas”, jugando por supuesto también con el contenido de la tarjeta. Como ejemplo, si incluimos una imagen a modo de fondo (**ImageView**), y una etiqueta de texto superpuesta y alineada al borde inferior (`layout_gravity="bottom"`) con fondo negro algo traslúcido (por ejemplo `background="#8c000000"`) y un color y tamaño de texto adecuados, podríamos conseguir una tarjeta con el aspecto siguiente.





```
<android.support.v7.widget.CardView
    android:id="@+id/card1"
    android:layout_width="match_parent"
    android:layout_height="200dp"
    card_view:cardCornerRadius="6dp"
    card_view:cardElevation="10dp"
    card_view:cardUseCompatPadding="true" >

    <ImageView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:src="@drawable/city"
        android:scaleType="centerCrop"/>

    <TextView
        android:id="@+id/txt2"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:padding="10dp"
        android:text="@string/tarjeta_2"
        android:layout_gravity="bottom"
        android:background="#8c000000"
        android:textColor="#ffe3e3e3"
        android:textSize="30sp"
        android:textStyle="bold"/>

</android.support.v7.widget.CardView>
```

IMPLEMENTANDO LISTAS

Para visualizar elementos en una lista, Android ofrece diferentes implementaciones: Spinners, ListView, ListActivity, ListView personalizables, GridView y RecyclerView. Como el nuevo RecyclerView permite implementar y mejorar el ListView y GridView los vamos a pasar por alto comentando ligeramente el primero, si quieres más información sobre ellos podrás encontrarla en el tema que te pasaremos como anexo.

Todas estas implementaciones reciben los datos a través de un adaptador. El adaptador es asignado a la lista con el métodos `setAdapter()`.

Un adaptador hereda de la clase `BaseAdapter`, el propio lenguaje ofrece adaptadores sencillos o básicos como `ArrayAdapter`, `CursorAdapter` o `RecyclerView.Adapter`.

- *ArrayAdapter*. Es el más sencillo de todos los adaptadores, y provee de datos a un control de selección a partir de un array de objetos de cualquier tipo.
- *SimpleAdapter*. Se utiliza para mapear datos sobre los diferentes controles definidos en un fichero XML de layout.
- *SimpleCursorAdapter*. Se utiliza para mapear las columnas de un cursor sobre los diferentes elementos visuales contenidos en el control de selección.
- *RecyclerView.Adapter*. Es el tipo de adaptador que utiliza el componente `RecyclerView`.

Además de proveer de datos a los controles visuales, el adaptador también será responsable de generar a partir de estos datos las vistas específicas que se mostrarán dentro del control de selección. Por ejemplo, si cada elemento de una lista estuviera formado a su vez por una imagen y varias etiquetas, el responsable de generar y establecer el contenido de todos estos “sub-elementos” a partir de los datos será el propio adaptador.



Listas desplegables (Spinners)

Las listas desplegables en Android se llaman Spinner. El usuario selecciona de la lista emergente un elemento que es el seleccionado para el control.

El código XML de un Spinner vacío es el siguiente:

```
<Spinner android:id="@+id/CmbOpciones"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content" />
```

Los elementos que mostrará nuestro Spinner no se pueden indicar directamente, sino que es necesario crear una estructura de elementos que podamos asociar a éste en nuestro código. En primer lugar debemos definir un Array de Strings en Java:

```
final String[] semana= new String[]{"Lunes", "Martes", "Miércoles", "Jueves", "Viernes", "Sábado", "Domingo"};
```

A continuación debemos definir el adaptador, en nuestro caso un objeto ArrayAdapter, que será el objeto que enlazaremos con nuestro Spinner.

```
ArrayAdapter<String> adaptador =
    new ArrayAdapter<String>(this,
        android.R.layout.simple_spinner_item, semana);
```

Creemos el adaptador en sí, al que pasamos 3 parámetros:

1. El *contexto*, que normalmente será simplemente una referencia a la actividad donde se crea el adaptador.
2. El ID del *layout* sobre el que se mostrarán los datos del control. En este caso le pasamos el ID de un layout predefinido en Android (android.R.layout.simple_spinner_item), formado únicamente por un control textView, pero podríamos pasarle el ID de cualquier layout de nuestro proyecto con cualquier estructura y conjunto de controles, más adelante veremos cómo.
3. El *array* que contiene los datos a mostrar.

Un ejemplo completo podría ser el siguiente, crea un proyecto llamado Spinner y añade el siguiente código en la MainActivity, y no olvides añadir un spinner con id=spinner en el layout activity_main.xml:

```
String [] colores={"Rojo", "Verde", "Azul"};
ArrayAdapter<String> adaptador=new ArrayAdapter<String>(this, android.R.layout.simple_list_item_1, colores);
listaColores=(Spinner) findViewById(R.id.spinner);
listaColores.setAdapter(adaptador);
```

Nota ampliativa

Una alternativa a tener en cuenta, si los datos a mostrar en el control son estáticos, sería definir la lista de posibles valores como un recurso de tipo string-array. Para ello, primero crearíamos un nuevo fichero XML en la carpeta/res/values llamado por ejemplo valores_array.xml e incluiríamos en él los valores seleccionables de la siguiente forma:

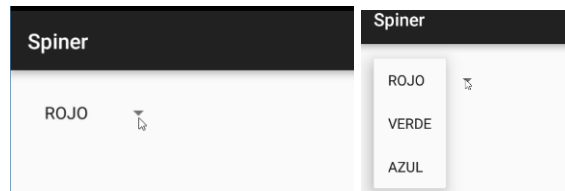
```
<string-array name="colores">
    <item>ROJO</item>
    <item>VERDE</item>
    <item>AZUL</item>
</string-array>
```



Tras esto, a la hora de crear el adaptador, utilizaríamos el método `createFromResource()` para hacer referencia a este array XML que acabamos de crear:

```
ArrayAdapter<CharSequence> adaptador=ArrayAdapter.  
    createFromResource(this,R.array.colores,android.R.layout.simple_list_item_1);
```

Con estas simples líneas tendremos un control similar al de la siguiente imagen:



En cuanto a los eventos lanzados por el control Spinner, el más utilizado será el generado al seleccionarse una opción de la lista desplegable, `onItemSelected`. Para capturar este evento se procederá de forma similar a lo ya visto para otros controles anteriormente, asignándole su controlador mediante el método `setOnItemSelectedListener()`, en este caso lo hacemos mediante el método de derivar de la interfaz:

```
        listaColores.setOnItemClickListener(this);  
  
@Override  
public void onItemSelected(AdapterView<?> adapterView, View view, int i, long l) {  
    Toast x=Toast.makeText(this,"El color elegido es: "+ colores[i], Toast.LENGTH_LONG);  
    x.show();  
}
```

Ejercicio Propuesto Spinner.

ListView

Un control ListView muestra al usuario una lista de opciones seleccionables directamente sobre el propio control, sin listas emergentes como en el caso del control Spinner. En caso de existir más opciones de las que se pueden mostrar sobre el control se podrá por supuesto hacer scroll sobre la lista para acceder al resto de elementos.

Para empezar, veamos cómo podemos añadir un control ListView a nuestra interfaz de usuario:

```
<ListView  
    android:id="@+id/listView"  
    android:layout_width="wrap_content"  
    android:layout_height="match_parent"/>
```

Necesitamos crear un adaptador como en el caso del Spinner y asignarlo al objeto lista que tenemos vinculado en el layout.



```
String[] datos;
ListView lista;
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    datos=new String[]{"lunes", "martes", "miercoles", "jueves", "viernes"};
    lista=(ListView) findViewById(R.id.listView);
    ArrayAdapter adapter=new ArrayAdapter(this,android.R.layout.simple_list_item_1,datos);
    lista.setAdapter(adapter);
}
```

El método que tenemos que tener en cuenta es el siguiente:

```
lista.setOnItemClickListener(new AdapterView.OnItemClickListener() {
    @Override
    public void onItemClick(AdapterView<?> adapterView, View view, int i, long l) {
        Toast.makeText(MainActivity.this, "Has pulsado el elemento" + i, Toast.LENGTH_SHORT).show();
    }
});
```

Ejercicio resuelto PoblacionPaísesAmericanos

ListActivity

La clase ListActivity hereda de la clase Activity y se diseñó con la finalidad de simplificar el manejo de *ListView*. *Solamente se deberá añadir al layout de la aplicación, un ListView con un id que se debe llamar @android:id/list* (id definido por el sistema, que permitirá referenciar automáticamente a la lista). Con este paso ya no tendremos que referenciar a la lista en ningún momento.

```
<ListView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:id="@android:id/list"></ListView>
```

El adaptador se asigna a la lista a través de `setListAdapter()`, mientras que con el `ListView`, se hace con `nombreLista.setAdapter()`.

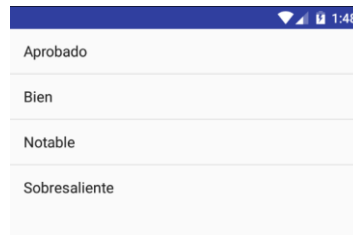
ListActivity no requiere, tampoco, que tengamos que asignar un layout a nuestra activity principal con el método `setContentView`. Por lo que el main quedaría:

```
public class MainActivity extends ListActivity {
    ArrayList<String> datos;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        datos=new ArrayList<>();
        datos.add("Aprobado"); datos.add("Bien");
        datos.add("Notable"); datos.add("Sobresaliente");
        ArrayAdapter adapter=new ArrayAdapter(this,android.R.layout.simple_list_item_1,datos);
        setListAdapter(adapter);
    }
}
```

Por defecto, se asigna el escuchador `onItemClickListener()` asociado al método `onListItemClick()`, por lo que no tendremos que aplicar el método `setOnItemClickListener()`.

```
@Override
protected void onListItemClick(ListView l, View v, int position, long id) {
    super.onListItemClick(l, v, position, id);
    Toast.makeText(this, "Has pulsado" + position, Toast.LENGTH_SHORT).show();
}
```





RecyclerView

Con la llegada de Android 5.0, Google ha incorporado al SDK de Android un nuevo componente que viene a mejorar a los clásicos ListView y GridView. RecyclerView se va a sustentar sobre otros componentes complementarios para determinar cómo acceder a los datos y cómo mostrarlos. Los más importantes serán los siguientes:

- RecyclerView.Adapter
- RecyclerView.ViewHolder
- LayoutManager
- ItemDecoration
- ItemAnimator

De igual forma que hemos hecho con los componentes anteriores, un RecyclerView se apoyará también en un adaptador para trabajar con nuestros datos, en este caso un adaptador que herede de la clase RecyclerView.Adapter. La peculiaridad en esta ocasión es que este tipo de adaptador nos “obligará” en cierta medida a utilizar un RecyclerView.ViewHolder (elemento que permite optimizar el acceso a los datos del adaptador).

Una vista de tipo RecyclerView por el contrario no determina por sí sola la forma en que se van a mostrar en pantalla los elementos de nuestra colección, sino que va a delegar esa tarea a otro componente llamado LayoutManager, que también tendremos que crear y asociar al RecyclerView para su correcto funcionamiento. Por suerte, el SDK incorpora de serie tres LayoutManager para las tres representaciones más habituales: lista vertical u horizontal (LinearLayoutManager), tabla tradicional (GridLayoutManager) y tabla apilada o de celdas no alineadas (StaggeredGridLayoutManager). Por tanto, siempre que optemos por alguna de estas distribuciones de elementos no tendremos que crear nuestro propio LayoutManager personalizado, aunque por supuesto nada nos impide hacerlo.

Los dos últimos componentes de la lista se encargarán de definir cómo se representarán algunos aspectos visuales concretos de nuestra colección de datos (más allá de la distribución definida por el LayoutManager), por ejemplo marcadores o separadores de elementos, y de cómo se animarán los elementos al realizarse determinadas acciones sobre la colección, por ejemplo al añadir o eliminar elementos.

Como hemos comentado, no siempre será obligatorio implementar todos estos componentes para hacer uso de un RecyclerView. Lo más habitual será implementar el Adapter y el ViewHolder, utilizar alguno de los LayoutManager predefinidos, y sólo en caso de necesidad crear los Item Decoration e Item Animator necesarios para dar un toque de personalización especial a nuestra aplicación.



Vamos a abrir el proyecto EjemploLista en el que teníamos diseñado nuestro CardView. Debemos añadir la dependencia a la librería de `supportrecyclerview-v7`, lo que nos permitirá el uso del componente RecyclerView en la aplicación:

Tras esto ya podremos añadir un nuevo RecyclerView al layout de nuestra actividad principal:

```
<android.support.v7.widget.RecyclerView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:id="@+id/recycler"/>
```

Como siguiente paso escribiremos nuestro adaptador. Este adaptador deberá extender la clase `RecyclerView.Adapter`, de la cual tendremos que sobrescribir principalmente tres métodos:

- `onCreateViewHolder()`. Encargado de crear los nuevos objetos `ViewHolder` necesarios para los elementos de la colección.
- `onBindViewHolder()`. Encargado de actualizar los datos de un `ViewHolder` ya existente.
- `onItemCount()`. Indica el número de elementos de la colección de datos.

Vamos a crear una clase nueva en nuestro proyecto que derive de `RecyclerView.Adapter` y sobrescribiremos los métodos que sean necesarios, quedando el código como vemos en la imagen. Posteriormente se explicará que se ha hecho en estos métodos.

```
public class Adaptador extends RecyclerView.Adapter {
    Context context;
    Holder holder;
    public Adaptador(Context context) {
        this.context=context;
    }
    @Override
    public RecyclerView.ViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {
        View view= LayoutInflater.from(parent.getContext()).inflate(R.layout.cardview,parent,false);
        return null;
    }
    @Override
    public void onBindViewHolder(RecyclerView.ViewHolder hd, int position) {

    }
    @Override
    public int getItemCount() {
        return 0;
    }
}
```

Como se puede ver en la imagen, en el método `onCreateViewHolder`, lo primero que se hace es inflar un elemento `View` usando el layout que habremos redefinido en los recursos, y en el que habremos definido el estilo que queremos que tenga cada elemento de nuestra lista. Para ello utilizaremos el layout que tenemos comenzado con la `CardView` y lo modificamos de la siguiente forma:

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.v7.widget.CardView xmlns:android="http://schemas.a
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:id="@+id/cardview"
    app:cardElevation="2dp"
    app:cardUseCompatPadding="true"
    app:cardCornerRadius="2dp"
    app:cardBackgroundColor="@android:color/holo_green_light">
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="horizontal">
    </LinearLayout
```




```

        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:gravity="center_vertical"
        android:paddingLeft="5dp"
        android:paddingRight="5dp"
        android:layout_weight="9"
        android:orientation="vertical">
        <TextView
            android:id="@+id/textonombre"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:textColor="@android:color/white"
            android:textStyle="bold"
            android:textSize="15dp"
            android:text="Name" />
        <TextView
            android:id="@+id/textapellido"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:textColor="@android:color/white"
            android:textSize="15dp"
            android:text="Apellidos" />
    </LinearLayout>
    <ImageButton
        android:id="@+id/imageButton"
        android:backgroundTint="@android:color/transparent"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:src="@android:drawable/ic_menu_share" />
</LinearLayout>
</android.support.v7.widget.CardView>

```

El siguiente paso para implementar el adaptador, será definir una clase que nos facilite el trabajo con los datos que se manipularán en nuestro Adaptador, en el ejemplo la clase Usuarios.

```

public class Usuario {
    String nombre;
    String apellidos;
    String correo;

    public Usuario(String nombre, String apellidos, String correo) {
        this.nombre = nombre;
        this.apellidos = apellidos;
        this.correo = correo;
    }

    public String getNombre() {
        return nombre;
    }

    public String getApellidos() {
        return apellidos;
    }

    public String getCorreo() {
        return correo;
    }

    public void setNombre(String nombre) {
        this.nombre = nombre;
    }

    public void setApellidos(String apellidos) {
        this.apellidos = apellidos;
    }

    public void setCorreo(String correo) {
        this.correo = correo;
    }
}

```

Después necesitaremos construir el ViewHolder. El patrón **ViewHolder** tiene un único propósito en su implementación: **mejorar el rendimiento**. Y es que cuando hablamos de aplicaciones móviles, el rendimiento es un tema de gran importancia. Lo que hacemos con el ViewHolder es mantener una referencia a los elementos de nuestro RecyclerView mientras el usuario realiza scrolling en nuestra



aplicación. Así que cada vez que obtenemos la vista de un ítem, evitamos las frecuentes llamadas a `findViewById`, esta se realizaría únicamente la primera vez y el resto llamaríamos a la referencia en el `ViewHolder`, ahorrándonos procesamiento.

Lo definiremos como otra clase extendida de la clase `RecyclerView.ViewHolder`, y será bastante sencillo, tan sólo tendremos que incluir como atributos las referencias a los controles del layout de un elemento de la lista (en nuestro caso los dos `TextView`) e inicializarlas en el constructor utilizando como siempre el método `findViewById()` sobre la vista recibida como parámetro. Por comodidad añadiremos también un método auxiliar, que llamaremos `bind()`, que se encargue de asignar los contenidos de los dos cuadros de texto a partir de un objeto `Usuarios` cuando nos haga falta.

```
public class Holder extends RecyclerView.ViewHolder {
    TextView txtNombre, txtApellido;

    public Holder(View itemView) {
        super(itemView);
        txtNombre=(TextView)itemView.findViewById(R.id.textonombre);
        txtApellido=(TextView)itemView.findViewById(R.id.textapellido);
    }

    public void bind(Usuario usuario)
    {
        txtNombre.setText(usuario.getNombre());
        txtApellido.setText(usuario.getApellidos());
    }
}
```

Finalizado nuestro `ViewHolder` podemos ya seguir con la implementación del adaptador sobrescribiendo los métodos indicados. En el método `onCreateViewHolder()` nos limitaremos a inflar una vista a partir del layout correspondiente a los elementos de la lista (`listitem_titular`), y crear y devolver un nuevo `ViewHolder` llamando al constructor de nuestra clase `Holder` pasándole dicha vista como parámetro.

```
public class Adaptador extends RecyclerView.Adapter {
    Context context;
    Holder holder;

    public Adaptador(Context context) {
        this.context=context;
    }

    @Override
    public RecyclerView.ViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {
        View view= LayoutInflater.from(parent.getContext()).inflate(R.layout.cardview,parent,false);
        holder=new Holder(view);
        return holder;
    }

    @Override
    public void onBindViewHolder(RecyclerView.ViewHolder hd, int position) {
        ((Holder)hd).bind(((MainActivity)context).datos[position]);
    }

    @Override
    public int getItemCount() {
        return ((MainActivity)context).datos.length;
    }
}
```

Los dos métodos restantes son aún más sencillos. En `onBindViewHolder()` tan sólo tendremos que recuperar el objeto correspondiente a la posición recibida como parámetro y llamar al método `bind` de nuestro `Holder`, pero siempre desde el objeto `ViewHolder` recibido como parámetro. Por su parte, `getItemCount()` tan sólo devolverá el tamaño de la lista de datos.

Con esto tendríamos finalizado el adaptador, por lo que ya podríamos asignarlo al `RecyclerView` en nuestra actividad principal. Tan sólo tendremos que crear nuestro adaptador personalizado, pasando como parámetros la lista de datos y asignarle al control `RecyclerView` mediante `setAdapter()`.



Tras obtener la referencia al RecyclerView se ha incluido también una llamada a `setHasFixedSize()`. Aunque esto no es obligatorio, sí es conveniente hacerlo cuando tengamos certeza de que el tamaño de nuestro RecyclerView no va a variar (por ejemplo debido a cambios en el contenido del adaptador), ya que esto permitirá aplicar determinadas optimizaciones sobre el control.

```
recycler=(RecyclerView) findViewById(R.id.recycler);
adaptador=new Adaptador(this);
recycler.setAdapter(adaptador);
```

El siguiente paso obligatorio será asociar al RecyclerView un LayoutManager determinado, para obtener la forma en la que se distribuirán los datos en pantalla. Como ya dijimos, si nuestra intención es mostrar los datos en forma de lista o tabla (al estilo de los antiguos ListView o GridView) no tendremos que implementar nuestro propio LayoutManager. En nuestro caso particular queremos mostrar los datos en forma de lista con desplazamiento vertical. Para ello tenemos disponible la clase LinearLayoutManager, por lo que tan sólo tendremos que instanciar un objeto de dicha clase indicando en el constructor la orientación del desplazamiento (LinearLayoutManager.VERTICAL o LinearLayoutManager.HORIZONTAL) y lo asignaremos al RecyclerView mediante el método `setLayoutManager()`. Esto lo haremos justo después del código anterior, tras la asignación del adaptador:

```
recycler.setHasFixedSize(true);
recycler.setLayoutManager(new LinearLayoutManager(this, LinearLayoutManager.VERTICAL, false));
```

El código completo de la MainActivity del proyecto EjemploLista podría ser algo parecido a lo siguiente:

```
public class MainActivity extends AppCompatActivity {
    Usuario[] datos=new Usuario []{new Usuario("Juan Pedro","Gomez Garcia","junape@hotmail.com"),
                                    new Usuario("Lola","Perez Pardo","lolitaperez@gmail.com"),
                                    new Usuario("Manuel","Garcia Rodrigez","manolitogafotas@hotmail.es")};

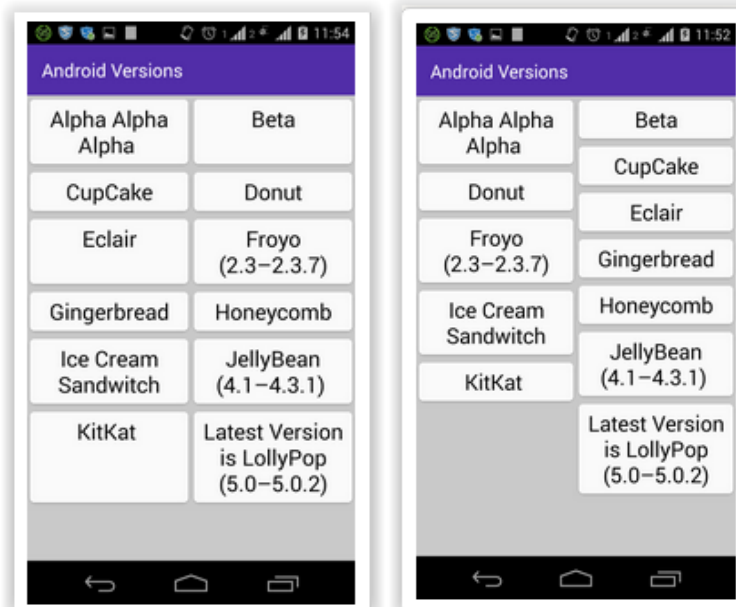
    RecyclerView recycler;
    Adaptador adaptador;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        recycler=(RecyclerView) findViewById(R.id.recycler);
        adaptador=new Adaptador(this);
        recycler.setAdapter(adaptador);
        recycler.setHasFixedSize(true);
        recycler.setLayoutManager(new LinearLayoutManager(this, LinearLayoutManager.VERTICAL, false));
    }
}
```

Llegados aquí ya sería posible ejecutar la aplicación de ejemplo para ver cómo quedan nuestros datos en pantalla, ya que los dos componentes que nos faltan por comentar (ItemDecoration e ItemAnimator) no son obligatorios para el funcionamiento básico del control RecyclerView.



GridLayoutManager y StaggeredGridLayoutManager

Lo bueno de todo lo que llevamos hasta el momento, es que si cambiáramos de idea y quisiéramos mostrar los datos de forma tabular tan sólo tendríamos que cambiar la asignación del LayoutManager anterior y utilizar un GridLayoutManager, al que pasaremos como parámetro el número de columnas a mostrar y si lo que queremos es mostrarla de forma tabular pero sin igualar las celdas usaremos el StaggeredGridLayoutManager, el tercer argumento dispone si tiene que dimensionar las celdas a su contenido o no.



```
recyclerView.setLayoutManager(new GridLayoutManager(this, 2));
```

```
recyclerView.setLayoutManager(new StaggeredGridLayoutManager(2,1));
```

El siguiente paso que nos podemos plantear es cómo responder a los eventos que se produzcan sobre el RecyclerView, como opción más habitual el evento click sobre un elemento de la lista. Para sorpresa de todos la clase RecyclerView no tiene incluye un evento onItemClick() como ocurría en el caso de ListView. Una vez más, RecyclerView delegará también esta tarea a otro componente, en este caso a la propia vista que conforma cada elemento de la colección, es decir al adaptador.

Aprovecharemos la creación de cada nuevo ViewHolder para asignar a su vista asociada el evento onClick. Adicionalmente, para poder hacer esto desde fuera del adaptador, incluiremos el listener correspondiente como atributo del adaptador, y dentro de éste nos limitaremos a asignar el evento a la vista del nuevo ViewHolder y a lanzarlo cuando sea necesario desde el método onClick(). Creo que es más fácil verlo sobre el código:



```

public class Adaptador extends RecyclerView.Adapter implements View.OnClickListener {
    Context context;
    Holder holder;
    View.OnClickListener listener;

    public Adaptador(Context context) {
        this.context=context;
    }

    @Override
    public RecyclerView.ViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {
        View view= LayoutInflater.from(parent.getContext()).inflate(R.layout.cardview,parent,false);
        holder=new Holder(view);
        view.setOnClickListener(this);
        return holder;
    }

    @Override
    public void onBindViewHolder(RecyclerView.ViewHolder hd, int position) {
        ((Holder)hd).bind(((MainActivity)context).datos[position]);
    }

    @Override
    public int getItemCount() {
        return ((MainActivity)context).datos.length;
    }

    public void setClickListener(View.OnClickListener listener)
    {
        if(listener!=null) this.listener=listener;
    }

    @Override
    public void onClick(View view) {
        if(listener!=null) listener.onClick(view);
    }
}

```

Como vemos, nuestro adaptador implementará la interfaz `OnClickListener`, declarará un listener (el que podremos asignar posteriormente desde fuera del adaptador) como atributo de la clase, en el momento de crear el nuevo `ViewHolder` asociará el evento a la vista, y por último implementará el evento `onClick`, que se limitará a lanzar el mismo evento sobre el listener externo. El método adicional `setClickListener()` nos servirá para asociar el listener real a nuestro adaptador en el momento de crearlo.

Veamos cómo quedaría nuestra actividad principal con este cambio:

```

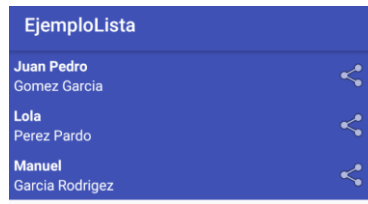
adaptador=new Adaptador(this);
adaptador.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        int pos=recycler.getChildAdapterPosition(view);
        Toast.makeText(MainActivity.this,datos[pos].getCorreo(),Toast.LENGTH_LONG).show();
    }
});
recycler.setAdapter(adaptador);

```

Por último vamos a describir brevemente los dos componentes restantes relacionados con `RecyclerView`: `ItemDecoration` e `ItemAnimation`

Si utilizamos `RecyclerView` con un layout básico como un `ListView` en vez de `CardView`, podremos ver que la salida por pantalla de nuestra lista ha perdido, como mínimo, el separador entre ítems. Esto ocurre porque los `RecyclerView` no aportan ningún formato, tendríamos que diseñarlo implementado las clases `ItemDecoration` e `ItemAnimation`.





Los **ItemDecoration** nos servirán para personalizar el aspecto de un RecyclerView más allá de la distribución de los elementos en pantalla. El ejemplo típico de esto son los separadores o divisores de una lista. RecyclerView no tiene ninguna propiedad divider como en el caso del ListView, por lo que dicha funcionalidad debemos suplirla con un ItemDecoration.

Crear un ItemDecoration personalizado no es una tarea demasiado fácil, o al menos no inmediata, por lo que por ahora no nos detendremos mucho en ello. Para el caso de los separadores de lista podemos encontrar en un ejemplo del propio SDK de Android un ejemplo de ItemDecoration que lo implementa https://android.googlesource.com/platform/development/+/_master/samples/Support7Demos/src/com/example/android/supportv7/widget/decorator/DividerItemDecoration.java.

La clase en cuestión se llama DividerItemDecoration y tendríamos que incluirla en el proyecto. Si estudiamos un poco el código veremos que tendremos que extender de la clase RecyclerView.ItemDecoration e implementar sus métodos getItemOffsets() , onDraw() y onDrawOver(). El primero de ellos se encargará de definir los límites del elemento de la lista y el segundo de dibujar el elemento de personalización en sí. En el fondo no es complicado, aunque sí lleva su trabajo construirlo desde cero.


Para utilizar este componente deberemos simplemente crear el objeto y asociarlo a nuestro RecyclerView mediante addItemDecoration() en nuestra actividad principal:

```
recyclerView.addItemDecoration( new DividerItemDecoration(this,DividerItemDecoration.VERTICAL_LIST));
```

Como nota adicional indicar que podremos añadir a un RecyclerView tantos ItemDecoration como queramos, que se aplicarán en el mismo orden que se hayan añadido con addItemDecoration().

Por último hablemos muy brevemente de **ItemAnimator**. Un componente ItemAnimator nos permitirá definir las animaciones que mostrará nuestro RecyclerView al realizar las acciones más comunes sobre un elemento (añadir, eliminar, mover, modificar). Este componente tampoco es sencillo de implementar, pero por suerte el SDK también proporciona una implementación por defecto que puede servirnos en la mayoría de ocasiones, aunque por supuesto podremos personalizar creando nuestro propio ItemAnimator. Esta implementación por defecto se llama DefaultItemAnimator y podemos asignarla al RecyclerView mediante el método setItemAnimator().

```
recyclerView.setItemAnimator(new DefaultItemAnimator());
```

 **EjercicioPropuesto, añade la funcionalidad del Click largo para eliminar el elemento pulsado de la lista. Para actualizar el adaptador una vez eliminado puedes usar adaptador.notifyItemRemoved(posicioneliminada)**



Click en cualquier lugar de la vista

Si quisiéramos detectar la pulsación de cualquier elemento de la línea del recycler, deberemos actuar sobre esa vista en el Holder (es donde podemos hacer referencia a cada uno de los view del layout). La forma de hacerlo es igual como se ha hecho anteriormente, usando un listener de la interfaz que nos haga falta y mandando la información a través de esta. En este caso, podría ser algo parecido a lo siguiente:

```
public class Holder extends RecyclerView.ViewHolder implements View.OnClickListener {
    TextView txtNombre, txtApellido;
    ImageButton imgBtn;
    View.OnClickListener listener;

    public Holder(View itemView) {
        super(itemView);
        txtNombre=(TextView) itemView.findViewById(R.id.textonombre);
        txtApellido=(TextView) itemView.findViewById(R.id.textapellido);
        imgBtn=(ImageButton) itemView.findViewById(R.id.imagebutton);
        imgBtn.setOnClickListener(this);
    }

    public void bind(Usuario usuario)
    {...}
    public void setClickBtImagen(View.OnClickListener listener)
    {
        if(listener!=null) this.listener=listener;
    }

    @Override
    public void onClick(View view) {
        if(listener!=null) listener.onClick(view);
    }
}
```

Como se puede ver en la imagen del código, se ha puesto un escuchador de Click corto en el elemento imagenButton, del layout cardview.xml, que todavía no habíamos usado.

En la clase adaptador deberemos llamar al método del Holder setClickListener (método que sirve para asignar los listener entre las clases), como vemos en la imagen. También tendremos que crear el método setClickBtImagen para poder seguir mandando la información hacia la Main.

```
public class Adaptador extends RecyclerView.Adapter implements View.OnClickListener, View.OnLongClickListener {
    Context context;
    Holder holder;
    View.OnClickListener listener, listenerBtImagen;
    View.OnLongClickListener longListener;

    public Adaptador(Context context) { this.context=context; }
    @Override
    public RecyclerView.ViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {
        View view= LayoutInflater.from(parent.getContext()).inflate(R.layout.cardview,parent,false);
        holder=new Holder(view);
        view.setOnClickListener(this);
        view.setOnLongClickListener(this);
        holder.setClickBtImagen((view) -> {
            if(listenerBtImagen!=null) listenerBtImagen.onClick(view);
        });
        return holder;
    }

    public void setClickBtImagen(View.OnClickListener listener)
    {
        if(listener!=null) listenerBtImagen=listener;
    }
}
```

En la MainActivity, tendremos que recoger la pulsación a través del adaptador llamando al método de este, como hemos hecho en los demás claso.:



```

adaptador.setClickBtImagen(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        Toast.makeText(MainActivity.this, "Pulsada Imagen", Toast.LENGTH_SHORT).show();
    }
});

```

Implementar interfaz para enviar información

Como podemos suponer, la View que estamos mandando hacia atrás no es la línea completa del recycler sino que es la vista imagen, por lo que no tendremos la información del elemento pulsado para acceder al array de datos. Para poder tener esa información en la MainActivity tendremos que crearnos un Interfaz propia con los argumentos que necesitemos mandar. La interfaz en nuestro caso podría ser:

```

public interface onImagenClickListener {
    void onImagenClick(Usuario usuario);
}

```

El resto de código quedará de la misma manera, sustituyendo la implementación del `OnClickListener` por nuestra propia interfaz. El Holder quedaría de la siguiente forma:

```

public class Holder extends RecyclerView.ViewHolder implements View.OnClickListener{
    TextView txtNombre, txtApellido;
    ImageButton imgBtn;
    onImagenClickListener listener;
    Usuario usuario;

    public Holder(View itemView) {
        super(itemView);
        txtNombre=(TextView)itemView.findViewById(R.id.textonombre);
        txtApellido=(TextView)itemView.findViewById(R.id.textapellido);
        imgBtn=(ImageButton)itemView.findViewById(R.id.imageButton);
        imgBtn.setOnClickListener(this);
    }

    public void bind(Usuario usuario)
    {
        txtNombre.setText(usuario.getNombre());
        txtApellido.setText(usuario.getApellidos());
        this.usuario =usuario;
    }

    public void setClickBtImagen(onImagenClickListener listener)
    {
        if(listener!=null) this.listener=listener;
    }

    @Override
    public void onClick(View view) {
        if(listener!=null) listener.onImagenClick(usuario);
    }
}

```



La parte afectada del adaptador quedaría de la siguiente manera

```
View.OnLongClickListener longListener;
onImagenClickListener listenerBtImagen;

public Adaptador(Context context) { this.context=context; }
@Override
public RecyclerView.ViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {
    View view= LayoutInflater.from(parent.getContext()).inflate(R.layout.cardview,parent,false);
    holder=new Holder(view);
    view.setOnClickListener(this);
    view.setOnLongClickListener(this);
    holder.setClickBtImagen(new onImagenClickListener() {
        @Override
        public void onImagenClick(Usuario usuario) {
            listenerBtImagen.onImagenClick(usuario);
        }
    });
    return holder;
}

public void setClickBtImagen(onImagenClickListener listener)
{
    if(listener!=null) listenerBtImagen=listener;
}
```

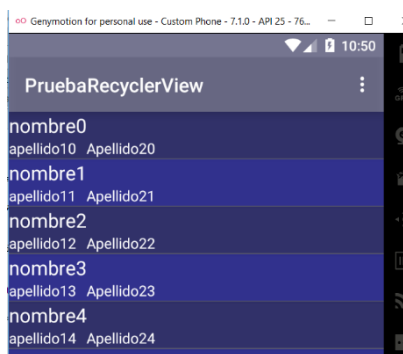
Y en la Main el método al que llamaríamos sería el siguiente:

```
adaptador.setClickBtImagen(new onImagenClickListener() {
    @Override
    public void onImagenClick(Usuario usuario) {
        Intent intento=new Intent(Intent.ACTION_SEND);
        intento.setType("text/html");
        intento.setData(Uri.parse("mailto:"));
        intento.putExtra(Intent.EXTRA_EMAIL, usuario.getCorreo());
        startActivity(Intent.createChooser(intento, "Email "));
        Toast.makeText(MainActivity.this,"Enviado Email a"+ usuario.getCorreo(), Toast.LENGTH_SHORT).show();
    }
});
```

Otros efectos sobre el Recycler

Cambiar color de líneas

Actuando sobre el Holder podemos conseguir otro tipo de efectos, como por ejemplo cambiar el color de las líneas pares y de las impares de la lista.



```
Context context;
public Holder(View v, Context context) {
    super(v);
    this.v=v;
    this.context=context;
    txtNombre = (TextView) v.findViewById(R
```

Estos efectos los manejaremos en el método bind, que es donde se carga cada una de las líneas de la lista. Tendremos que pasar como parámetro al método o incluso en el constructor del Holder, los argumentos necesarios. En este caso pos, que sería el número de línea que está cargando y el context de MainActivity que nos puede hacer falta para acceder a los recursos lo pasaríamos en el constructor del Holder (imagen de código superior). Para que el context lo podamos pasar al constructor del Holder, antes tendremos que haberselo pasado al constructor del Adaptador.

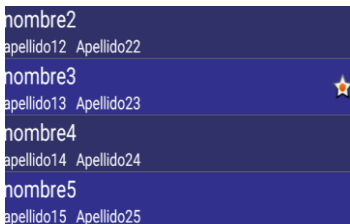


```
public void bind(Usuarios entity, int pos){
    if(pos%2==0)v.setBackgroundColor(ContextCompat.getColor(context,R.color.oscuro));
    else v.setBackgroundColor(ContextCompat.getColor(context,R.color.claro ));
    txtNombre.setText(entity.getNombre());
    txtApellido.setText(entity.getApellidos());
}
```

Hacer visible algún elemento

Por otro lado podríamos conseguir el efecto de añadir algún elemento si se cumple determinada condición por ejemplo, en este caso lo que hacemos es mostrar una imagen, que en principio está oculta en el layout de la línea del recycler, en el caso de que el nombre contenga un 3.

El código también tendría que ir en el bind del Holder



```
public void bind(Usuarios entity, int pos){
    if(pos%2==0)v.setBackgroundColor(ContextCompat.getColor(context,R.color.oscuro));
    else v.setBackgroundColor(ContextCompat.getColor(context,R.color.claro ));
    if(entity.getNombre().contains("3")) estrella.setVisibility(View.VISIBLE);
    else estrella.setVisibility(View.INVISIBLE);
    txtNombre.setText(entity.getNombre());
    txtApellido.setText(entity.getApellidos());
}
```

Detectar swipe izq/der sobre un elemento del recycler

Una opción muy utilizada en las listas, es la de controlar el deslizamiento a la izquierda o hacia la derecha sobre uno de sus elementos. Para ello tendremos que usar una clase que ya está definida y que tendremos que copiar en nuestro proyecto, llamada SwipeDetector (se pasa en los recursos). En el Adaptador tendríamos que implementar la interfaz onTouch, para que nos detecte el desplazamiento, esto lo haremos como lo hicimos en puntos anteriores.

```
class Adaptador extends RecyclerView.Adapter implements View.OnClickListener,
    View.OnTouchListener, View.OnLongClickListener{

    private ArrayList<Usuarios> datos;
    private View.OnClickListener listener;
    private View.OnTouchListener listenerTouch;
    private View.OnLongClickListener listenerLong;
    private Context context;
    private Adaptador(ArrayList<Usuarios> datos, Context context) {...}
    @Override
    public RecyclerView.ViewHolder onCreateViewHolder(ViewGroup viewGroup, int i) {
        View itemView = LayoutInflater.from(viewGroup.getContext())
            .inflate(R.layout.recyclerlayout, viewGroup, false);
        itemView.setOnClickListener(this);
        itemView.setOnTouchListener(this);
        itemView.setOnLongClickListener(this);
        return new Holder(itemView, context);
    }
    //ONTOUCH
    void setOnTouch(View.OnTouchListener listener) { this.listenerTouch=listener; }
    @Override
    public boolean onTouch(View v, MotionEvent event) {
        if(listenerTouch!=null)
            listenerTouch.onTouch(v, event);
        return false;
    }
}
```

En la ActivityMain, tendremos que llamar al método setOnTouch del adaptador pasándole un objeto de la clase SwipeDetector, pero a la hora de detectar el movimiento tendremos que usar la interfaz onClikListener, como vemos en la imagen. Si nos fijamos en el código, podremos ver que el objeto swipeDetector nos sirve para detectar si el movimiento se ha producido a la derecha o a la izquierda.



```

swipeDetector=new SwipeDetector();
adaptador.setOnTouch(swipeDetector);
adaptador.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        if (swipeDetector.swipeDetected()) {
            if (swipeDetector.getAction() == SwipeDetector.Action.LR) {
                Toast.makeText(MainActivity.this, "Derecha", Toast.LENGTH_SHORT).show();
            } else if (swipeDetector.getAction() == SwipeDetector.Action.RL) {
                Toast.makeText(MainActivity.this, "Izquierda", Toast.LENGTH_SHORT).show();
            }
        } else
            Toast.makeText(MainActivity.this, "Has pulsado" + recyclerView.getChildAdapterPosition(v)
                , Toast.LENGTH_SHORT).show();
    }
});

```

-  **Ejercicio Resuelto FiltradoDeListas**
-  **Ejercicio Propuesto Agenda**

